# **Module5:** AWT Applet and Swings

Date __/__/__

# AWT Controls

## "Introduction to AWT"

→ AWT contains numerous classes and methods that allow to create and manage windows.

→ AWT effectively used to create applets or stand alone GUI based application

→ AWT provides windows, GUI controls (labels, buttons, list) , frames, layout managers etc.

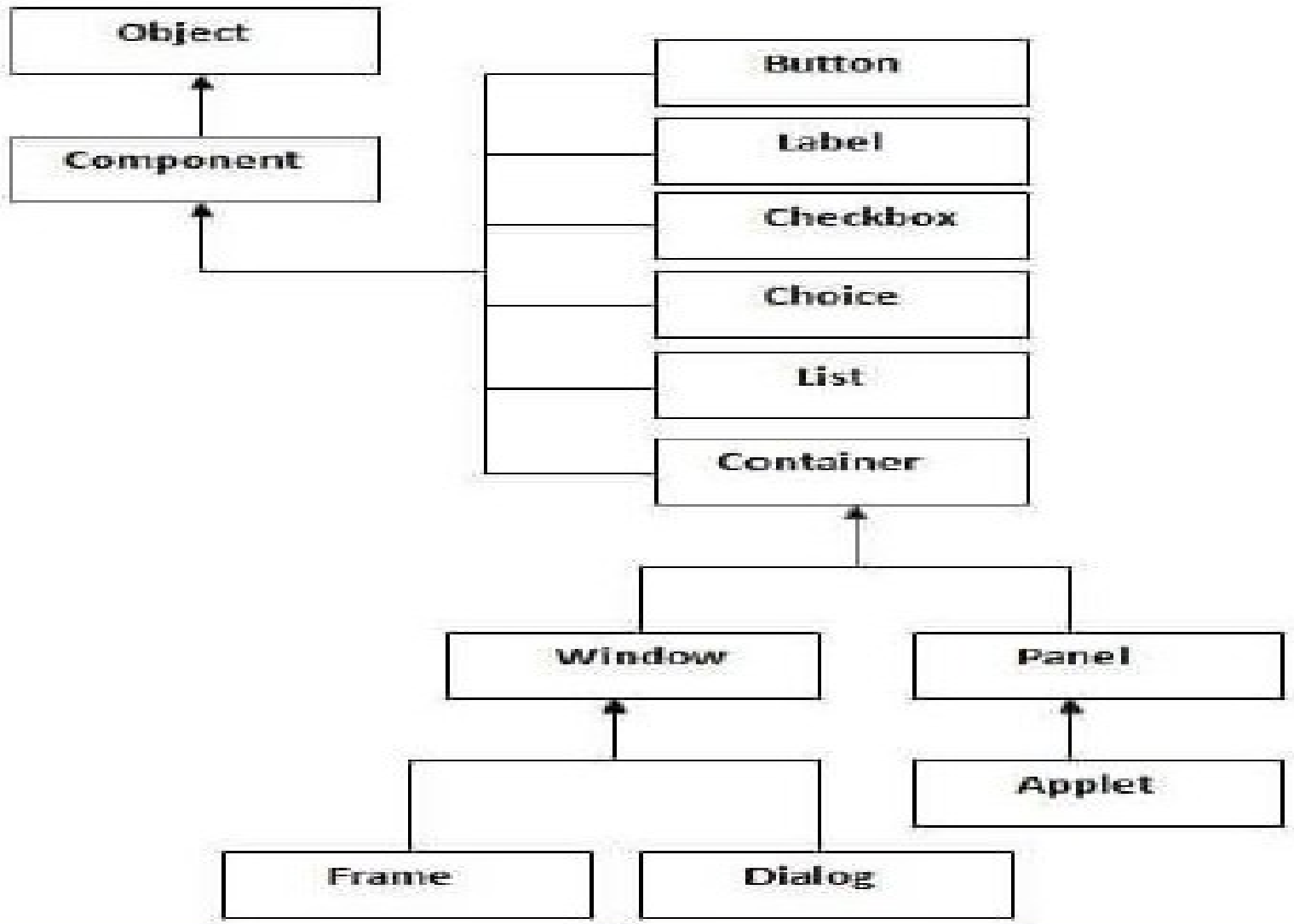
→ AWT provides the class to work with text, graphics, colors etc

→ AWT is the Basis upon which Swing is Built.

→ The applets uses AWT to provide GUI.

# AWT Hierarchy

The hierarchy of Java AWT classes are given below.



# AWT Classes

- **The AWT classes are contained in the java.awt package.**
- It is one of Java's largest packages.
- **It is logically organized in a top-down, hierarchical fashion.**

| Class Name | Description |
|---|---|
| BorderLayout | The border layout manager. Border layouts use five components: North, South, East, West, and Center. |
| Button | Creates a push button control. |
| Canvas | A blank, semantics-free window. |
| CardLayout | The card layout manager. Card layouts emulate index cards. Only the one on top is showing |
| Checkbox | Creates a check box control. |
| CheckboxGroup | CheckboxGroup |
| Choice | Creates a pop-up list. |
| Color | Manages colors in a portable, platform-independent fashion. |
| Component | An abstract super class for various AWT components. |
| Container | A subclass of Component that can hold other components. |
| FileDialog | Creates a window from which a file can be selected. |
| FlowLayout | The flow layout manager. Flow layout positions components left to right, top to bottom. |
| Font | Encapsulates a type font. |

| | |
|---|---|
| Frame | Creates a standard window that has a title bar, resize corners, and a menu bar. |
| Graphics | Encapsulates the graphics context. This context is used by the various output methods to display output in a window. |
| GridBagConstraints | Defines various constraints relating to the GridBagLayout class. |
| GridBagLayout | The grid bag layout manager. Grid bag layout displays components subject to the constraints specified by GridBagConstraints. |
| GridLayout | The grid layout manager. Grid layout displays components in a twodimensional grid. |
| Image | Encapsulates graphical images. |
| Label | Label Creates a label that displays a string. |
| List | Creates a list from which the user can choose. Similar to the standard Windows list box. |
| Menu | Creates a pull-down menu. |
| MenuBar | Creates a menu bar. |
| Menultem | Creates a menu item. |
| Panel | The simplest concrete subclass of Container. |
| Point | Encapsulates a Cartesian coordinate pair, stored in x and y. |
| Polygon | Encapsulates a polygon. |
| Rectangle | Encapsulates a rectangle. |
| Scrollbar | Creates a scroll bar control. |
| TextArea | Creates a multiline edit control. |
| TextField | Creates a single-line edit control. |
| Window | Creates a window with no frame, no menu bar, and no title. |

# Window Fundamentals

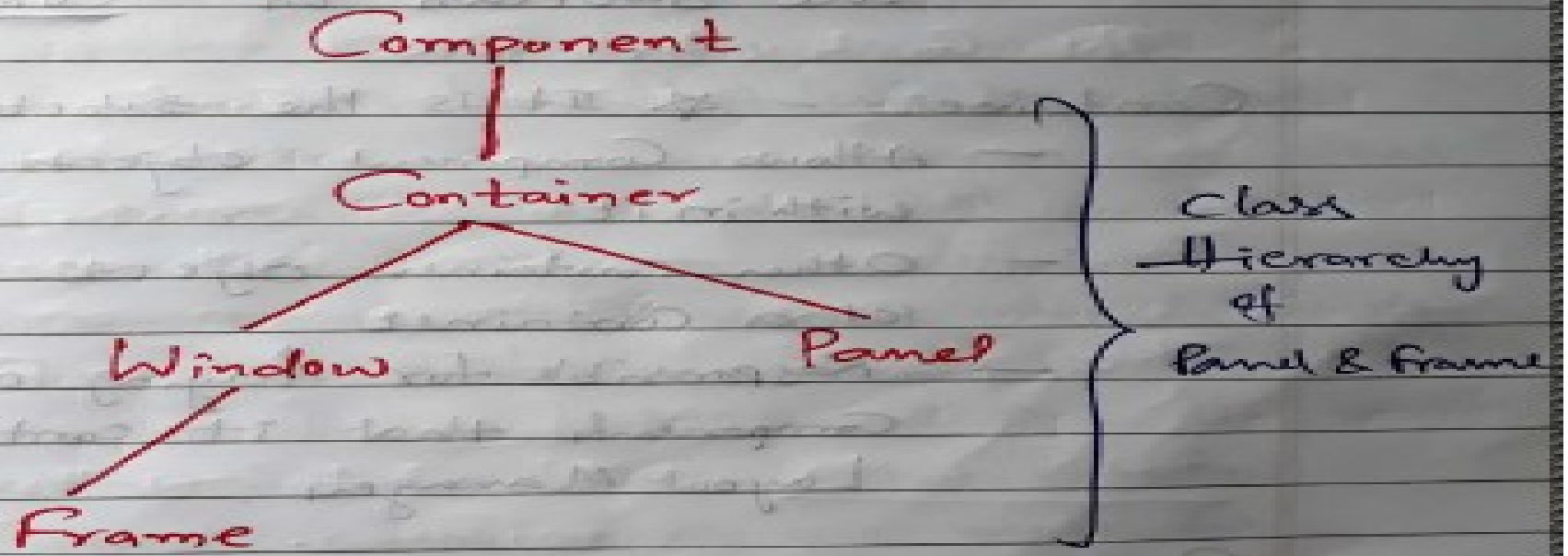
Date ___ / ___ / ___

Gaathi

AWT defines Windows for GUI

Windows are derived from

1. Panel — used by applets
2. Frame — Creates a Standard application window




Component —
- Top of AWT Hierarchy
- Abstract class that has all attributes of a visual component.
- defines several methods for handling events

Page No.

* Responsible for remembering the current foreground & background colors, currently selected text font.
* The the GUI tools except menus are subclass of "Component".

Container – * It is the subclass of Component
— Allows Component objects to be nested with in it
— Other Container objects stored inside of a Container
— Responsible for laying out any Components that it contains, using Layout Managers.

Panel — It is a subclass of Container
— It simply implements Container
— It is superclass for Applet
— A Panel is a window that does not contain a title bar, menu bar or border.

— When applet is Run on browser all the bars and borders are not shown
— Applet viewer from the title and borders.
— Components are added to Panel object by using add() method.

Window — Creates a top level window, it is not contained within any other object.
— It directly on the desktop.
— We can not directly create window objects
— Its subclass Frame is used.

Frame — It is a subclass of Window.
— It has title bar, menu bars, borders, Resizing corners

Canvas — It is a blank window upon which we can draw.
— It is not the part of hierarchy

# Working with ~~Graphics~~ Frame Windows

Date ____ / ____ / ____

Saathi

- A Frame allows to create window
- Used for stand alone applications.
  [Ex: Calculator, Word file etc].
- Allows to create child windows in Applets.

The Constructors are

Frame ( ) – Creates a standard window without title

Frame (String title) – Creates a window with specified title.

The methods defined in Frame class are

setSize( ) – Used to set the dimensions of window.

void setSize( int wid, int hgt)
void setSize( Dimension size)

wid & hgt are specified in pixels.

Dimension getSize( ) – returns the current size of window contained in width & height fields of Dimension.

setVisible() — Once a frame window is created, it will not be visible directly, until calls setVisible().

void setVisible(boolean flag)

flag → true ⇒ frame visible

flag → false ⇒ frame is Hidden.

setTitle() — To change or title or set the title of frame window

void setTitle(String title).

To close the window, setVisible(false) is called in close event

```
import java.awt.*;
class MyFrame extends Frame
{
    MyFrame (String title)
    {
        super(title);
    }
}
public class FrameDemo
{
    public static void main (String args[])
    {
        MyFrame f= new MyFrame
                    ("Basic Frame");
        f.setSize(300,400);
        f.setVisible(true);

        // f.setTitle("Hello frames");
    }
}
```

```
public void Paint
{ Graphics g)
g.drawString("Hi"
x,y);
}
```

# Control Fundamentals

Date ___ / ___ / ___

Saathi

- In AWT, Controls are components that allow a user to interact with the application (Program).
- A layout manager automatically positions components within a container.
- Appearance of window is combination of Controls and layout manager used.
- AWT Supports the following controls.
  - Labels
  - Push Buttons
  - Check boxes, Checkbox Group,
  - Choice lists.
  - Lists
  - Scroll Bars
  - Text Area, Layout Managers,
- Adding Controls

add() method of Component class is used to add controls to the window

Component add (Component obj)

obj — Component instance.
i.e. instance of control

Page No.

- Removing Controls

  remove() method of Container class is used to remove a control from the window if no longer needed.

  void remove(Component obj)

  obj — instance of control to remove.

  void removeAll() — used to remove all controls on window.

- Most of the Controls in AWT Cause throw HeadlessException if non-Interactive application is used.

- Responding to Controls is possible by using Event Handling mechanism.

The AWT Controls are

1. **Labels** –

- A label is an object of type Label class
- A Label contains a String to display.
- Labels are passive controls, have no interaction with the user.

Constructors are
- Label() → empty label
- Label(String str)
- Label(String str, int how)

str — String to be displayed
how — the alignment of label.
(Label.LEFT, Label.RIGHT, Label.CENTER)

- The methods defined are

void setText(String str) – to Specify label
String getText() – to obtain the label
void setAlignment(int how) — To set the alignment of label.
int getAlignment() – Returns the alignment used for the label.

In init() method of
Date ___/___/___ ↓ Constructor of Frame. (or) Saathi

```
Label l1 = new Label();
l1.setLabel("CSE");
Label l2 = new Label("ECE");
Label l3 = new Label("EEE", Label.RIGHT);
add(l1);
add(l2);
add(l3);
```

By default, if no layout manager is used, AWT uses Default layout manager which places from top left corner onwards.

Date ___/___/_____

Saathi

2. **Buttons** —

- A push down button is a component that contains a label
- Generates an event when it is pressed.
- There are objects of type "Button" class
- The Constructors are
  - Button() — empty button
  - Button(String str) — labeled button
- The methods defined are
  - void setLabel(String str)
  - String getLabel()
  - str — is the label for the button.
- Each time a button is pressed, an action event is generated.

- ActionEvent — ActionListener —
  - actionPerformed(),
  - getActionCommand().

```
Button b1 = new Button();
Button b2 = new Button("yes");
add(b1); add(b2);
```

Page No.

3. Checkbox

- A check box (GUI) is used to turn an option on or off.
- It contains a small box that can either contains a check mark or not.
- A label with each each check box tells what option it represents.
- To change state, click on the box
- Checkboxes can be used in group or individual
- checkboxes are objects of Checkbox class
- The constructors are

```
Checkbox()
Checkbox(String str)
Checkbox(String str, boolean on)
Checkbox(String str, boolean on, CheckboxGroup g)
Checkbox(String str, CheckboxGroup g, boolean on)
```

* if on is true then the checkbox is initially checked (✓).
* g is null if Checkbox is not belongs to a group.

The methods defined are

boolean getState() –

void setState (boolean on) –

String getLabel() –

Void setLabel (String str) –

ItemEvent – ItemListener –
itemStateChanged()

Eg:

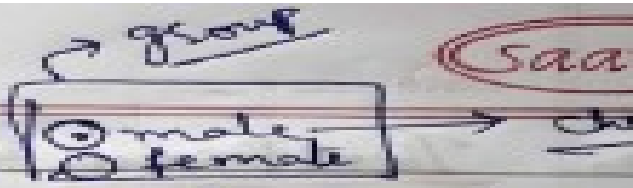
```
Checkbox c1 = new Checkbox ("ECE", tru
```

↳ ☑ ECE

Date __/__/____

Saathi

4. CheckboxGroup




- A set of mutually exclusive check boxes in which only one check box in the group can be checked at a time, called Radio buttons [on radio only one channel selected].
- A checkbox group is created by using the class CheckboxGroup class object.
- The Constructor is
  CheckboxGroup()
- The methods defined are
  Checkbox getSelectedCheckbox()
  ↳ Returns the selected checkbox in group
  void setSelectedCheckbox(Checkbox which)
  ↳ to select the specified checkbox.

ItemEvent — ItemListener — itemStateChanged()

Eg:

```
CheckboxGroup g = new CheckboxGroup();
Checkbox c1 = new Checkbox("male", g);
Checkbox c2 = new Checkbox("female", g);
```

Date ___/___/___ Saathi

5. TextField

String
Hello

- Single line text area is created by using Object of the class TextField.
- called "edit Control".
- Used to enter & edit text
- The Constructors are
  - TextField ( ) — default
  - TextField ( int noChars) — no. of chars wide
  - TextField ( String str)
  - TextField (String str, int noChars)

str — to keep initial text in the field

- The methods defined are
  - String getText() — Returns the string currently contained in text field.
  - void setText(String str) — To set or assign text in the text field
  - String getSelectedText() — Returns the string selected in text field
  - void select(int start, int end) — Selects the text from start to end.

void setEditable ( boolean flag) — if flag is true, then allows to change text

boolean isEditable ( ) — returns if the text is changed in text field.

void setEchoChar ( char ch) — used for passwords. 'ch' is which char is for echoing

Ex: t1. setEchoChar ('*');

char getEchoChar ( ) — Returns the char used for echoing.

↓ By Adding a button to window

ActionEvent — ActionListener — actionPerformed ( )

```
TextField t1 = new TextField("Hi");
t1.setEchoChar('*');
t1.setText("Hello");
```

Textfield → TextEvent → TextListener → textChanged ( ).

6. TextArea() —

- Used to edit multiple lines
- Called multiline editor
- The Constructors are

TextArea()
TextArea(int nolines, int nochars)
TextArea(String str)
TextArea(String str, int nolines, int nochars)
TextArea(String str, int nolines, int nochars, int sBars)

nolines → no. of ed Rows
nochars → no of Chars in a row (width)
Str → Initial text
sBars → Scroll bars.

The values can be

SCROLLBARS_BOTH
SCROLLBARS_NONE
SCROLLBARS_VERTICAL_ONLY
SCROLLBARS_HORIZONTAL_ONLY

/ The methods defined are

getText()
setText()
getSelectedText()
select()
isEditable()
setEditable()

} Same as TextField class

void append(String str) - appends the specified string at the end

void insert(String str, int index) - inserts the string at specified index.

Eg:

```
TextArea ta1 = new TextArea();
TextArea ta2 = new TextArea(10,15);
```

TextArea — TextEvent → TextListener → textChanged().

7. Choice Control

- Used to create a Pop-up list of items from which the user can select.
- Each item in the list is a String
- The class Choice is used to create the Control
- An item is added to the list by using add() method
  void add(String str).
- default Constructor Choice().
- The methods defined are

String getSelectedItem() — Returns the selected item from the list

int getSelectedIndex() — Returns the index of selected item (starts from 0).

int getItemCount() — Returns no. of items in the list.

String getItem(int index) —

void select(int index) —

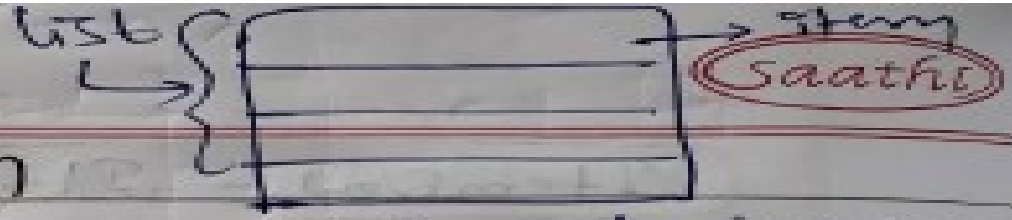
void select(String name) —

ItemEvent – ItemListener – itemStateChanged

Eg:

```
Choice c1 = new Choice();
c1.add("CSE");
c1.add("ECE");
```

Date ___ / ___ / ___




## 8. List Control

- The list class is used to create List Control
- It is a Compact, multiple-choice, Scrolling selection list.
- The Constructors are
  - List ( )
  - List (int noRows)
  - List (int noRows, boolean flag)

if flag is true allows to select multiple items

- To add a selection to the list add() method is used
  - void add (String Str)
  - void add (String str, int index)
- The methods defined are
  - String getSelectedItem()
  - int getSelectedIndex()
  - String[] getSelectedItems()
  - int[] getSelectedIndexes()

Page No.

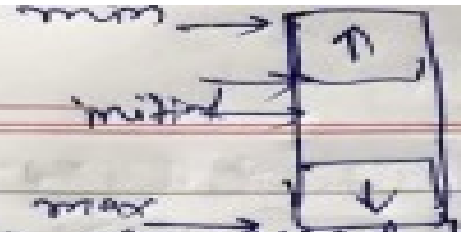
int getItemCount()
void select(int index)
String getItem(int index).

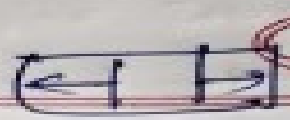
Eg:

```
List ls1 = new List(5, true);
ls1.add("CSE");
ls1.add("ECE");
ls1.add("EEE", 1);
Count = ls1.getItemCount();
```

List — ActionEvent — ActionListener
( — actionPerformed().
double click on item

List — ItemEvent — ItemListener
( — itemStateChanged().
item change



Saathi

Date ___ / ___ / ___

9. Scrollbar

- Scrollbars are used to select continuous values b/n specified minimum & maximum.
- Scrollbars may be oriented in horizontally or vertically.
- Each end has an arrow that we can click to move the ~~current value~~ Scrollbar.
- The current value of Scrollbar is indicated by thumb (or) slider box.
- Slider box can be dragged to a new position
- Scrollbars are created by using Scrollbar class.
- The Constructors defined are

  Scrollbar() throws HeadlessException
  Scrollbar(int style)
  Scrollbar(int style, int initialvalue,
  int thumbSize, int min, int max)

The default constructor creates a vertical Scroll bar

"style" can take → Scrollbar.Horizontal
→ Scrollbar.Vertical

for first two constructors to set the parameters, setValues() is called.

```
Void setValues( int initialvalue, int thumbSize,
                int min, int max)
```

The methods defined are

```
int getValue()
Void setValue( int val)
int getMinimum()
int getMaximum()
void setUnitIncrement ( int val)
Void setBlockIncrement ( int val).
```

Eg:

```
Scrollbar s1 = new Scrollbar
        (Scrollbar.vertical, 0, 1, 0, height);
Scrollbar s2 = new Scrollbar
        (Scrollbar.HORIZONTAL, 0, 1, 0, width);
add(s1);
add(s2);
```

Scrollbar → Adjustment Event → Adjustment Listener → adjustmentValueChanged()

```
import java.awt.*;
import java.awt.event.*;
public class LoginDemo extends
Frame implements ActionListener
{
Label l1,l2,l3;
TextField t1,t2;
Button b1;
LoginDemo(String name)
{
super(name);
l1=new Label("User Name");
l1.setBounds(20,50,100,20);
//x,y -> top left point , width,height
are dimensions
l2=new Label("Password");
l2.setBounds(20,100,100,20);
t1= new TextField();
t1.setBounds(130,50,100,20);
t2= new TextField();
t2.setBounds(130,100,100,20);
t2.setEchoChar('*');
b1= new Button("login");
b1.setBounds(80,150,80,20);
l3 = new Label("->");
l3.setBounds(80,200,200,20);
add(l1);add(t1);
add(l2);add(t2);
add(b1);add(l3);
b1.addActionListener(this);
setSize(400,400);
setLayout(null);
setVisible(true);
/* addWindowListener(new
WindowAdapter(){
public void windowClosing(WindowEvent
e) {
dispose();
```

```
public void actionPerformed(ActionEvent
e)
{
String uname = t1.getText();
String pwd = t2.getText();
if(uname.equals("vce") &&
pwd.equals("root"))
l3.setText("Welcome to VCE");
else
l3.setText("Invalid Username or
Password");
}
public static void main(String[] args)
{
new LoginDemo("Login Window");
//fRAME NAME IS lOGIN wINDOW
}
}
```

```
/* Frame Window - ItemEvent*/
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
public class PersonalDemo extends
Frame implements ItemListener
{
Label l1,l2,l3,l4,l5,l6;
TextField t1;
Checkbox m,f,c1,c2,c3;
CheckboxGroup g1;
Choice ch;
List ls;
TextArea ta;
String msg;
public PersonalDemo ()
{ setLayout(new FlowLayout(5,5));
setSize(600,600);
setVisible(true);
l1 = new Label("Name");
t1 = new TextField(25);
l2 = new Label(" Gender ");
g1 = new CheckboxGroup( );
m = new Checkbox(" Male " , g1,true);
f = new Checkbox(" Female " ,
g1,false);
l3= new Label(" Property ");
c1 = new Checkbox("Home");
c2 = new Checkbox("Car");
c3 = new Checkbox("Bike");
l4 = new Label(" Residing City ");
ch = new Choice(); ch.add("Hyderabad");
ch.add("Bangalore"); ch.add("Delhi");
ch.add("Mumbai");
```

```
add(l2);add(m);add(f);
add(l3);add(c1);add(c2);add(c3);
add(l4);add(ch); add(l5);add(ls);
add(l6);add(ta);
m.addItemListener(this);
f.addItemListener(this);
c1.addItemListener(this);
c2.addItemListener(this);
c3.addItemListener(this);
ch.addItemListener(this);
ls.addItemListener(this);
}
public void paint(Graphics g)
{
Font f = new Font("Dailog" ,
Font.BOLD ,15);
g.setFont(f);
g.setColor(Color.red);
g.drawString("Personal Profile" ,
35,490);
}
```

```
public void itemStateChanged(ItemEvent
ie)
{
String name = t1.getText();
String gender =
g1.getSelectedCheckbox().getLabel();
String prop="";
if(c1.getState() )
prop = prop +c1.getLabel() + " " ;
if (c2.getState())
prop = prop +c2.getLabel() + " ";
if (c3.getState())
prop = prop +c3.getLabel();
String city = ch.getSelectedItem() +
" ";
```

# Layout Managers

Date ____ / ____ / ____

Saathi

- A Layout manager automatically arranges user controls within a window by using an algorithm.
- Positioning manually the controls on a large window is more difficult.
- Java provides Layout managers to position the controls.
- Container object has a layout manager associated with it
- A Layout manager is an instance of any class that implements "LayoutManager" interface.
- If no layout manager is used, Java provides default layout managers.

A Layout manager is set by using
void setLayout( LayoutManager obj)

→ to set manually then obj is null.

Each Layout manager keeps track of a list of components and stored by their names.

Java has several predifined LayoutManager classes

1. FlowLayout
2. BorderLayout
3. GridLayout
4. CardLayout
5. GridBagLayout.

# 1. FlowLayout

It is the default layout manager
It is a Simple layout Style (like text editor).
Layout Components from Left to right, top to bottom. (Line by line from upper left corner).
The Constructors are
FlowLayout()
FlowLayout(int how)
FlowLayout(int how, int hor, int ver)

① The default constructors, centers the Components and leave 5 pixels of space between each Component.

② How each line is aligned is specified "how".
The values are
FlowLayout.LEFT
FlowLayout.CENTER
FlowLayout.RIGHT
FlowLayout.LEADING
FlowLayout.TRAILING

③ Horizontal & vertical Spacing between components is specified using hor & ver

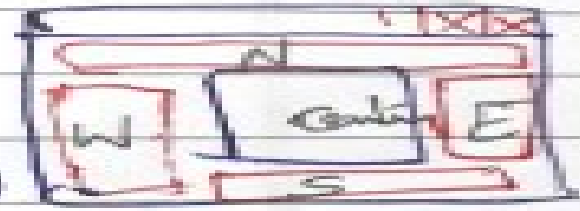
Ex: FlowLayoutDemo.java

```
//Program to Demonstrate
FlowLayout
import java.awt.*;
import java.awt.event.*;
public class FlowDemo extends
Frame implements ItemListener
{
String msg = "";
Checkbox c1,c2,c3,c4;
FlowDemo()
{
Label l1 = new Label("Languages
Known");
setVisible(true);
setSize(400,400);
setTitle("FlowLayout Example");
setBackground(Color.pink);
Color c = new Color(220,100,220);
setForeground(c);
Font f = new Font("Arial" ,Font.BOLD,10);
setFont(f);
FlowLayout fl = new
FlowLayout(FlowLayout.LEFT , 15,15);
setLayout(fl);
c1 = new Checkbox("C", true);
c2 = new Checkbox("C++");
c3 = new Checkbox("JAVA");
c4 = new Checkbox(".NET");
add(l1);
add(c1);add(c2);add(c3);add(c4);
c1.addItemListener(this);
c2.addItemListener(this);
c3.addItemListener(this);
c4.addItemListener(this);
/*addWindowListener(new
WindowAdapter()
```

```
public void
itemStateChanged(ItemEvent ie)
{
repaint();
}
public void paint(Graphics g)
{
msg = "Current Selection is: ";
g.drawString(msg, 6, 80);
msg = " C---> " + c1.getState();
g.drawString(msg, 6, 100);
msg = " C++---> " +c2.getState();
g.drawString(msg, 6, 120);
msg = "JAVA---->" + c3.getState();
g.drawString(msg, 6, 140);
msg = ".NET--->" + c4.getState();
g.drawString(msg, 6, 160);
}
public static void main(String args[])
{
new FlowDemo();
}
}
```

# 2. BorderLayout

- It is used for top-level windows
- The layout has four narrow, fixed-width components at the edges and one large area in the center.
- The four sides are called north, south, east & west.
- The Constructors are
  - BorderLayout ( )
  - BorderLayout (int hor, int ver)




hor, ver are used to specify the horizontal and vertical space left between components.

- The Constants defined in BorderLayout are
  - BorderLayout.CENTER
  - BorderLayout.EAST
  - BorderLayout.WEST
  - BorderLayout.SOUTH
  - BorderLayout.NORTH

To add Components with this layout add() method is used.

Void add(Component obj, Object region)

Eg: BorderLayoutDemo.java.

```
import java.awt.event.*;
import java.awt.*;
public class BorderDemo extends
Frame implements ActionListener
{
Button n,e,w;
Label s;
TextArea ta;
String msg="" ;
BorderDemo()
{
setVisible(true);
setSize(600,400);
BorderLayout bl = new
BorderLayout(10,10);
setLayout(bl);
n = new Button("CSM");
e = new Button("ECE");
w= new Button("EEE");
s = new Label("Border Layout");
ta = new TextArea(msg);

add(n,BorderLayout.NORTH);
add(s,BorderLayout.SOUTH);
add(e, BorderLayout.EAST);
add(w, BorderLayout.WEST);
add(ta, BorderLayout.CENTER);
n.addActionListener(this);
e.addActionListener(this);
w.addActionListener(this);
}
```

```
public void
actionPerformed(ActionEvent ae)
{
String str = ae.getActionCommand();
switch(str)
{
case "CSM" : msg= "CSM"; break;
case "ECE" : msg = "ECE";break;
case "EEE" : msg = "EEE"; break;
}
ta.setText(msg);
repaint();
}
public static void main(String args[])
{
new BorderDemo();
}
}
```

# 3. Grid Layout

Grid Layout lays out components in a two-dimensional grid.

The no. of rows & Columns are Specified while creating our object

The Constructors are

GridLayout() — Creates a Single column grid layout

GridLayout(int rows, int cols) — Creates a grid layout with the Specified no. of rows & cols.

GridLayout(int rows, int cols, int hor, int ver) — Creates a grid layout with the Specified no. of rows & cols, the spacing between components is specified using hor & ver.

rows = 0 ⇒ allows unlimited length cols

cols = 0 ⇒ allows unlimited length rows

```
import java.awt.*;
import java.applet.*;
public class GridDemo extends
Frame
{
static final int n = 4;
GridDemo()
{
setLayout(new GridLayout(n, n,10,10));
setVisible(true);
setSize(400,400);
setFont(new Font("SansSerif",
Font.ITALIC, 24));
for(int i = 0; i < n; i++)
{
for(int j = 0; j < n; j++)
{
int k = i * n + j;
add(new Button("" + k));
}
}
addWindowListener(new
WindowAdapter()
{
public void windowClosing(WindowEvent
e)
{
dispose();
}
});
}
public static void main(String args[])
{
new GridDemo();
```

# 4. Card Layout

- This Layout manager stores several different Layouts.
- Each layout can be thought of as being on a seperate index card in a deck, that can be shuffled so that any card is on top at a given time.
- The constructors are
  CardLayout() — creates a default card layout
  CardLayout(int hor, int ver) —
  — Allows to specify the horizontal & vertical space left b/n components.

- To create this Layout
  ① All individual cards are held in an object of type "Panel". [Cards]
  ② The cards that form the deck also Objects of type "Panel". [Main panel]
  ③ Add all the cards to the main "Panel" for which the "CardLayout" is the manager.

Page No.

④ Add this main Panel to the window

⑤ To access individual cards provide a button for each card.

To add panels to the main card (panel) deck
add() method is used.

Void add(Component panel, Object name)

↳ Each individual Panel is given a name.

The method defined by the CardLayout are

Void first(Container deck)

Void last(Container deck)

Void next(Container deck)

Void previous(Container deck)

Void show(Container deck, String card name)

↳ show() method displays the card whose name is passed as name.

```
import java.awt.*;
import java.awt.event.*;
public class CardDemo extends
Frame implements
ActionListener,MouseListener
{
Checkbox c1,c2,c3,c4; //B.Tech
Checkbox c5,c6,c7; //M.Tech
Panel deck; //deck
Panel p1,p2; //cards
CardLayout cl;
Button b1,b2;
public CardDemo()
{ setSize(400,400);
setVisible(true);
setLayout(new FlowLayout());
b1 = new Button("B.Tech");
b2 = new Button("M.Tech");
add(b1); //Frame window
add(b2); //Frame Window
cl = new CardLayout(5,5);
deck = new Panel();
deck.setLayout(cl); // set main panel
layout to card layout
c1 = new Checkbox("CSE", true);
c2 = new Checkbox("ECE");
c3 = new Checkbox("EEE");
c4 = new Checkbox("MECH");
p1 = new Panel(); //first card as
Panel
p1.add(c1);
p1.add(c2);
p1.add(c3);
p1.add(c4);
c5 = new Checkbox("Neural Networks",
true);
```

```
//Adding panel cards to main panel
deck.add(p1,"B.Tech");
deck.add(p2,"M.Tech");
// add main panel (deck) to Frame
add(deck);
b1.addActionListener(this);
b2.addActionListener(this);
addMouseListener(this);
}
public void mousePressed(MouseEvent
me)
{
cl.next(deck);
}
public void mouseClicked(MouseEvent
me)
{
}
public void
mouseEntered(MouseEvent me)
{
public void mouseExited(MouseEvent me)
{
cl.last(deck);
}
public void mouseReleased(MouseEvent
me)
{ }
public void actionPerformed(ActionEvent
ae)
{
if(ae.getSource() ==b1) //returns the
clicked button object
cl.show(deck, "B.Tech");
else
cl.show(deck, "M.Tech");
```

Unit - V

Date ___ / ___ / ___

# Applet Basics

Saathi

Java Applications

- Console Based Appl'ns [Keyboard, monitor]
- Applets

**Applets** – Applets are small applications that are accessed on an Internet server, transported over the internet, automatically installed and run as part of a web document.
- Applets provide a Graphical User Interface.
- Allows to develop Interactive applications.

A Simple Applet Example.

```
[import java.io.*;]
import java.awt.*;
import java.applet.*;

/*
<applet code = "AppletDemo" width=200 height=300>
</applet>
*/
```

Page No.

```
Public class AppletDemo extends Applet
{
    public void paint (Graphics g)
    {
        g.drawString ("Welcome to Applets", 20,20);
    }
}
```

An Applet can be run/executed as

① Using a standard tool appletviewer, it executes the applet in a window.

```
> javac AppletDemo.java        (Provided by JDK)
> appletviewer AppletDemo.java.
```

② Executing the applet within a Java-Compatible Web browser.

i.e. Step 1 : Create the source file in Java.

```
public class AppletDemo extends Applet
{
    public void paint (Graphics g)
    {
        g.drawString ("Welcome to Applets", 20,20);
    }
}
```

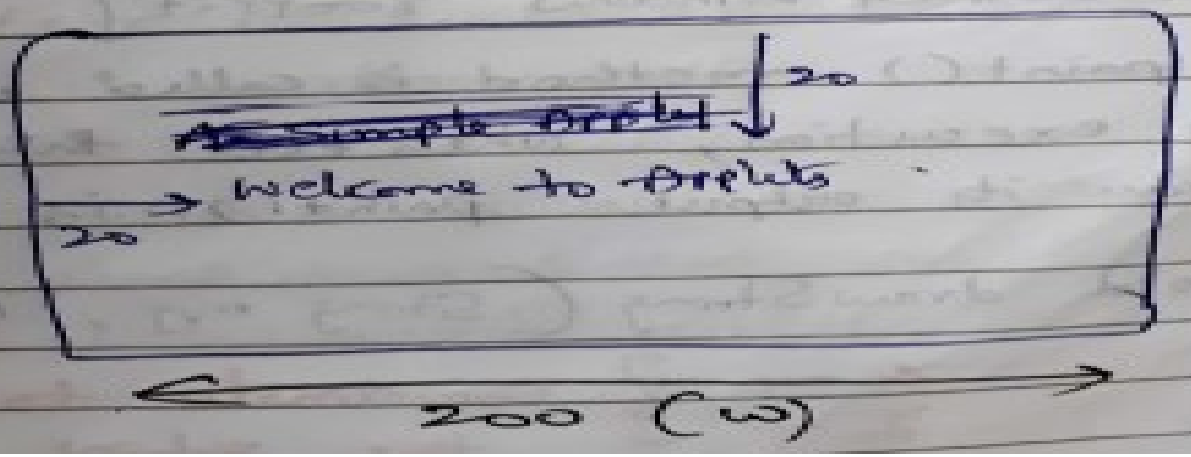
Step 2 :- Compile the Java Source file
Date ___/___/___ > javac AppletDemo.java.

Step 3: Create an HTML source file to run on browser

demo.html
```
<html>
  <body>
    <applet code = "AppletDemo.class"
            width = 200 height = 300
    </applet>
  </body>
</html>.
```

Step 4 : Open a web browser compatible with Java.

Address bar → E:\demo.html.



## The Key Points of an Applet are

1. Applets donot need a main() method
2. Applets must be run under an appletviewer or a Java-Compatible browser
3. User Input-output is performed with Java's Stream I/o classes. Applets use the Interface provided by AWT or Swing.
4. All applets are Subclasses of Applet class
5. Applets are not Stand alone programs, window based applications.

→ Applet class is defined in applet package

→ Graphics class is defined in awt package [Abstract Window Toolkit]

→ The paint() method is called when an applet begins execution, whenever the applet must Redraw its output paint() is called.

void drawString (String msg, int x, int y)

x — } Coordinates to display
y — } the String.

Page No.

The applets are of two types.

① Applet class — uses AWT to provide GUI

② JApplet class — Extends Applet, uses the Swing classes to Provide GUI. [inherits Applet].

Date ___/___/___

# Applet Lifecycle

Saathi

The life cycle of applet is managed by Java plug-in software using 5 methods.

**init() –**
- It is the first method to be called when an applet is executed.
- Used to initialize variables, create objects etc
- Called only once during the runtime of applet.

**Start() –**
- Called after init().
- to start or restart an applet this method is used.
- start() is called each time when an event occure.
- Called each time an Applet's HTML document is displayed on screen.

Date ___/___/___

paint() — Provided by AWT Components
Called each time when applet's output must be redrawn.
Called when applet begins execution
for every redrawn paint is called

stop() — Called when a browser leaves the HTML document containing the applet, when it goes to another page
we can restart by calling start() method.
Suspends execution.

destroy() — This method is called when the environment determining that applet needs to be removed completely from memory.
Any resources the applet using may be freed.
stop() method is always called before destroy.

update() — To update portion of a applet window

Default implementations for init(), start(), stop() & destroy() are provided by the Applet class.

Page No. ___

# Applet Skeleton

Saathi

```
import java.awt.*;
import java.applet.*;
/*
<applet code = "AppletCycle"
        width = 200 height = 200>
</applet>
*/

public class AppletCycle extends Applet
{
    // called first
    public void init()
    {
        // initialization
    }
// called after init().
// Also called when the applet is restarted.
    public void start()
    {
        // start or resume execution
    }
```

```
// called when the applet is stopped.
public void stop()
    {
        // Suspends execution
    }
// called when applet is terminated,
// this is the last method executed.
    public void destroy()
    {
        // Performs Shutdown activities.
    }
// Called when an applets window restored.
public void paint (Graphics g)
    {
        // Redisplay the contents of window
    }
}
```

When an applet begins

1. init()
2. start()
3. paint() called in sequence &

When an applet ends / terminated

1. stop()
2. destroy() } are called in the sequence.

# Simple Applet Display Methods.

Date ___ / ___ / ___

1. drawString() — To output a string on the applet. Defined in "Graphics" class
   void drawString(String msg, int x, int y)
   — output begins at (x, y).
   — Starts screen of applet (0,0) → Top left corner

2. setBackground() — To set back ground color of applets window.
   setForeground() — To set the foreground color like Color of text.
   — defined in Component class.
   void setBackground(Color col)
   void setForeground(Color col)
   - The Color class names can be
   Color.cyan, Color.magenta, Color.pink etc

   Color getBackground() — Returns the background color of Applet
   Color getForeground() — Returns the foreground color of Applet

Page No.

3. showStatus() — To output a message to the status window of the browser or applet viewer on which it is running.
— Used to report some error messages, suggest options etc.

void showStatus(String msg)

4. repaint() — It is defined in AWT. It causes the AWT runtime system to execute a call to applets update() interna paint() method.

void repaint() — Entire window repainted.

void repaint(int left, int top, int wid, int hgt) — The region specified is repainted.

void repaint(long maxdelay) — To repaint in a constant time interval.

void repaint(long maxDelay, int left, int top, int wid, int hgt)

// Example to use display methods.

```
import java.awt.*;
import java.applet.*;
/* <applet code = "AppletDemo2"
        width = 200 height = 500>
</applet> */

public class AppletDemo2 extends Applet
{
    String msg;
    public void init()
    {
        setBackground (Color.magenta);
        setForeground (Color.blue);
        msg = " init() called →";
    }
    public void start()
    {
        msg += " start() called →";
    }
```

Page No.

```
public void paint (Graphics g)
    {
        msg += " Inside paint()";
        J.drawString (msg, 20, 40);
        showStatus ("Applet display");
    }
}
```

Date ___/___/___

# The HTML APPLET Tag

Saathi

- Applet tag is used to manually start an applet.
- An Applet viewer will execute each APPLET tag
- A web browser can allow many applets on a single page.

The Syntax of APPLET tag is

```
< APPLET
  [CODEBASE = codebaseURL]
  CODE = appletFile
  [ALT = alternateText]
  [NAME = appletInstanceName]
    WIDTH = pixels   HEIGHT = pixels
  [ALIGN = alignment]
  [VSPACE = pixels] [HSPACE = pixels] >
[<PARAM NAME = attributeName VALUE = attValue>]
[<PARAM NAME = attributeName2 VALUE = attValue2>]
  =
</APPLET>
```

Page No. ____

Date ___/___/___

The Attributes are

CODEBASE — Specifies the URL of the applet code, which is the directory that will be searched for applet's executable class file. It is optional attribute.

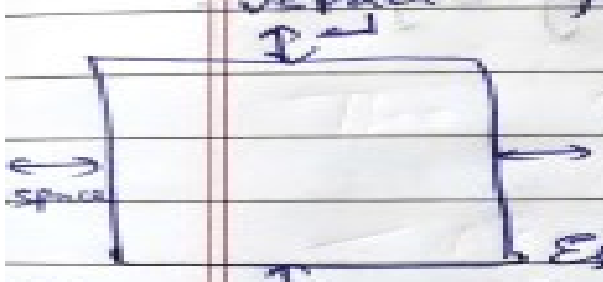
CODE — name of the file containing applet's compiled ".class" file It is required attribute

ALT — To specify a short text msg that should be displayed if the browser recognizes the APPLET tag but can't currently run Java applets. It is an optional attribute

NAME — Used to specify the name of the applet instance. It is an optional attribute. An applet must be named to distinguish different applets on same page.

WIDTH & HEIGHT — Gives the size of applet display area in pixels. These are required attributes

ALIGN — An optional attribute, Specifies the alignment of applet. The values are LEFT, RIGHT, TOP, BOTTOM, MIDDLE, BASCLINE, TEXTTOP, ABSMIDDLE, ABSBOTTOM.




Eg: ALIGN = "MIDDLE".

VSPACE — Specifies the space in pixels above & below the applet, optional attribute

HSPACE — Specifies the space in pixels on each side (left & right) of an applet. This attribute is optional.

PARAM NAME & VALUE — The tag <PARAM> allows to specify applet specific arguments. Applet acces their attributes with getParameter()

Date ____ / ____ / ____

# Passing Parameters to Applet

Saathi

- Parameters are passed to an applet by the <PARAM> tag inside the <APPLET> tag.
- The value of the parameter can be retrieved by using getParameter( ). It returns in the String object.

```
import java.awt.*;
import java.applet.*;
/*
<applet code = "ParamDemo" width = 300
                           height = 200>
<param name = fontName value = Courier>
<param name = fontSize value = 20>
<param name = opnd1 value = 20>
<param name = opnd2 value = 40>
</applet>
*/
```

```
public class ParamDemo extends Applet
{
    String name;
    int size, a, b, c;
    public void start()
    {
        name = getParameter("fontName");
        size = Integer.parseInt(getParameter("fontSize"));
        a = Integer.parseInt(getParameter("opnd1"));
        b = Integer.parseInt(getParameter("opnd2"));
        c = a+b;
    }

    public void paint(Graphics g)
    {
        g.drawString("font Name"+name, 10, 10);
        g.drawString("font size"+size, 10, 40);
        g.drawString("Addition"+c, 10, 70);
    }
}
```

paramname

Page No.

# Programs Using Applets

## 1. ActionEvent

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
<APPLET code = "AppEvent" width=200
height=200>
</APPLET>
*/
public class AppEvent extends Applet
implements ActionListener
{
Button b1;
Label l1;
TextField t1,t2;
public void init()
{
l1 = new Label("Enter");
t1 = new TextField(30);
t2= new TextField(30);
Button b1 = new Button("Click Me");
add(l1);
add(t1);
add(t2);
add(b1);
b1.addActionListener(this);
}
public void actionPerformed(ActionEvent
ae)
{
String str = t1.getText();
t2.setText(str);
t1.setText("");
repaint();
```

## 2. ActionEvent – Arithmetic Operations

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
<APPLET code="ArithDemo" width=250
height=250>
</APPLET>
*/
public class ArithDemo extends Applet
implements ActionListener
{ Button b1,b2,b3,b4,b5;
TextField t1,t2,t3 ;
Label l1,l2,l3;
public void init()
{ b1 = new Button("ADD");
b2 = new Button("SUB");
b3 = new Button("MUL");
b4 = new Button("DIV");
b5 = new Button("REM");
t1 = new TextField( 15);
t2 = new TextField( 15);
t3 = new TextField( 15);
l1 = new Label("Operand -1");
l2 = new Label("Operand -2");
l3 = new Label(" Result ");
add(l1);
add(t1);
add(l2);
add(t2);
add(l3);
add(t3);
add(b1);
add(b2);
add(b3);
add(b4);
add(b5);
b1.addActionListener(this);
b2.addActionListener(this);
b3.addActionListener(this);
b4.addActionListener(this);
b5.addActionListener(this);
}
```

```
public void paint(Graphics g)
{
Font f1 = new Font("Serif" , Font.BOLD ,
20);
g.setFont(f1);
g.setColor(Color.red);
g.drawString("Arithmetic Operations" ,
10,150);
}
public void actionPerformed(ActionEvent
ae)
{
String str= ae.getActionCommand();
//Returns label of button pressed
int a = Integer.parseInt(t1.getText());
int b = Integer.parseInt(t2.getText());
int c=0;
switch(str)
{
case "ADD" : c=a+b; break;
case "SUB" : c=a-b; break;
case "MUL" : c=a*b; break;
case "DIV" : c=a/b; break;
case "REM" : c=a%b; break;
}
t3.setText(" " +c+ " ");
repaint();
}
}
```

## 4. ItemEvent – Personal Profile

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/* <APPLET code="PersonalDemo" width=300
height=600>
</APPLET> */
public class PersonalDemo extends Applet
implements ItemListener
{
Label l1,l2,l3,l4,l5,l6;
TextField t1;
Checkbox m,f,c1,c2,c3;
CheckboxGroup g1;
Choice ch;
List ls;
TextArea ta;
String msg;
public void init()
{
l1 = new Label("Name"); t1 = new
TextField(25);
l2 = new Label(" Gender ");
g1 = new CheckboxGroup( );
m = new Checkbox(" Male " , g1,true);
f = new Checkbox(" Female " , g1,false);
l3= new Label(" Property ");
c1 = new Checkbox("Home");
c2 = new Checkbox("Car");
c3 = new Checkbox("Bike");
l4 = new Label(" Residing City ");
ch = new Choice();
ch.add("Hyderabad");
ch.add("Bangalore");
ch.add("Delhi");
ch.add("Mumbai");
l5 = new Label(" Do you Have ");
ls= new List(3,true);
```

```
add(l3);add(c1);add(c2);add(c3);
add(l4);add(ch); add(l5);add(ls);
add(l6);add(ta);
m.addItemListener(this);
f.addItemListener(this);
c1.addItemListener(this);
c2.addItemListener(this);
c3.addItemListener(this);
ch.addItemListener(this);
ls.addItemListener(this);
}
public void paint(Graphics g)
{
Font f = new Font("Dailog" , Font.BOLD ,15);
g.setFont(f);
g.setColor(Color.red);
g.drawString("Personal Profile" , 35,490);
}
```

```
public void itemStateChanged(ItemEvent ie)
{
String name = t1.getText();
String gender =
g1.getSelectedCheckbox().getLabel();
String prop="";
if(c1.getState() )
prop = prop +c1.getLabel() + " " ;
if (c2.getState())
prop = prop +c2.getLabel() + " ";
if (c3.getState())
prop = prop +c3.getLabel();
String city = ch.getSelectedItem() + " ";
String list[] = ls.getSelectedItems();
msg=name+ " : " + gender+ " : " + prop +
" : " + city;
```

## 5. KeyEvent

```
//Key events Demo
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
<applet code="KeyDemo" width=300
height=300>
</applet>
*/
public class KeyDemo extends Applet
implements KeyListener
{
String msg = "";
int X = 40, Y = 80; // output coordinates
public void init()
{
setForeground(Color.red);
addKeyListener(this);
}
public void keyPressed(KeyEvent ke)
{ showStatus("Key Down");
int key = ke.getKeyCode();
switch(key)
{
case KeyEvent.VK_F1 : msg +=
"<F1>";break;
case KeyEvent.VK_F2: msg +=
"<F2>";break;
case KeyEvent.VK_F3: msg +=
"<F3>";break;
case KeyEvent.VK_PAGE_DOWN:msg +=
"<PgDn>";break;
case KeyEvent.VK_PAGE_UP:msg +=
```

```
public void keyTyped(KeyEvent ke)
{
msg += ke.getKeyChar(); //gets the char stroked
repaint();
}
public void paint(Graphics g)
{
Color c1 = new Color(123,50,89);
g.setColor(c1);
Font f1 = new Font("Dailog" , Font.BOLD , 20);
g.setFont(f1);
g.drawString(msg, X, Y);
}
}
```

## 6. Mouseevents

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*<applet code="MouseDemo"
width=300 height=100>
</applet>*/
public class MouseDemo extends Applet
implements MouseListener,
MouseMotionListener
{
String msg = "";
int X = 0, Y = 0;
public void init()
{
addMouseListener(this);
addMouseMotionListener(this);
}
public void mouseClicked(MouseEvent
me)
{
X = 10;
Y = 10;
msg = "Mouse clicked.";
repaint();
}
public void mouseEntered(MouseEvent
me)
{ X = 0;
Y = 10;
```

```
public void mouseExited(MouseEvent
me)
{
X = 20;
Y = 100;
msg = "Mouse exited.";
repaint();
}
public void mousePressed(MouseEvent
me)
{
X = me.getX();
Y = me.getY();
msg = "Down";
repaint();
} public void
mouseReleased(MouseEvent me)
{
X = me.getX();
Y = me.getY();
msg = "Up";
```

```
public void mouseDragged(MouseEvent
me)
{
X = me.getX();
Y = me.getY();
msg = "*";
showStatus("Dragging mouse at " + X +
", " + Y);
repaint();
}
public void mouseMoved(MouseEvent
me)
{
```

# Swings

Saathi

## Introduction

Java swings is part of Java foundation classes (JFC) used to create window-based applications.

Swings are built on top of AWT API and completely written Java.

Swings is a set of classes that provides more powerful and flexible GUI components, Provides the look and feel for modern GUI.

The main package is "javax.swing".

## Drawbacks / Limitations of AWT

1. The various controls, windows provided by AWT are providing limited GUI.
2. The visual GUI components are platform specific. i.e. Look & feel of component is defined by the platform, but not Java.
3. AWT components uses native code resources, hence refered as "heavy weight".
4. The look & feel changes when diffent OS's browser run same application. This is not follows Java's "write once - run anywhere"
5. Controls are rectangular & Positions.

# Swing Features

Swing will resolve the limitations present in the AWT by using two key feature.

1. Light weight Components
2. Pluggable look and feel.

1. Swing components are light weight-

- Swing is written entirely in Java and do not map directly to platform-specific end Hence Swing Components are "Light weight"
- Provides non-rectangular shapes
- Light weight Components are more efficient and flexible.
- The look & feel is determined by swing but not operating system.
- Each Component will work in a Consistent (Same) manner across all platforms.

② Swing Supports Pluggable look & feel.

- look & feel of Component is under the Control of Swing.
- Changing of Component how to present is easily managed.
- It is possible to "plug-in" a new look and feel for any given Component without any side effects.

③ Supports MVC architecture

- Swing supports Model-View-Controller architecture, it is a web application development model.

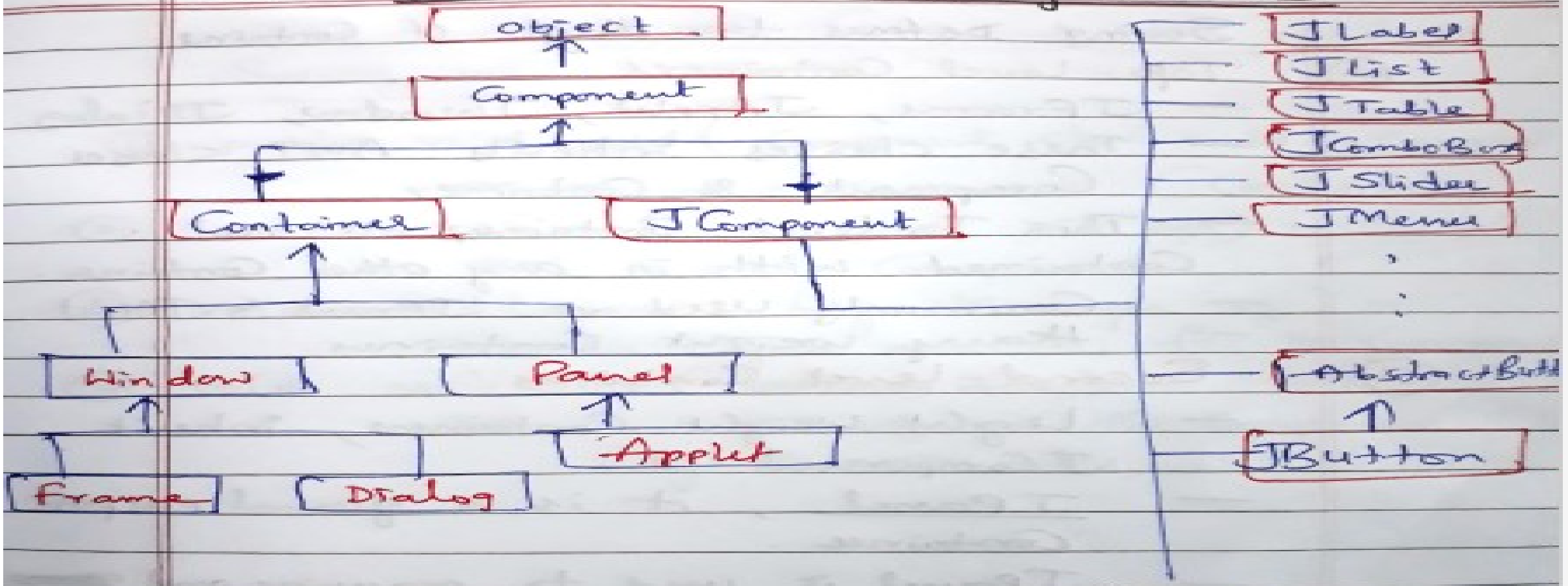
# Difference b/n AWT & Swing.

| AWT | Swing |
|---|---|
| ① AWT Components are platform dependant | ① Swing Components are platform-independant |
| ② AWT Components are heavy weight | ② Swing Components are light weight. |
| ③ AWT does not Support pluggable look & feel. | ③ Swing Supports the pluggable look and feel. |
| ④ AWT Provides less no.of Components than Swing | ④ Swing Provides more Powerful Components like tables, lists, Scroll panes, tabbed pane etc. |
| ⑤ AWT does not follows MVC architecture | ⑤ Swing follows the MVC – Model Architecture. |
| ⑥ Less control on the Controls | ⑥ More flexible & Control on the Controls. |

# Hierarchy of Java Swing classes

Date ___/___/___

Sgathi




- All Swing GUIs will have atleast one Container. Containers are Components and hence a Container can have other Containers. This Swing forms "Containment hierarchy".
- Swing Containers are JFrame, JApplet, JWindow, JPanel etc.

# Top Level Containers

Date ___ / ___ / ___

Saathi

Swing defines two types of Containers.

1. Top-Level Containers
   - JFrame, JApplet, JWindow, JDialog.
   - These classes inherits AWT classes Component & Container.
   - This Top level Containers are not Contained with in any other Container.
   - Commonly used are JFrame & JApplet.
   - Heavy weight Containers.
2. Second-Level Containers
   - Light weight Containers, inherit JComponent.
   - JPanel, it is a general purpose Container.
   - JPanel is used to organize and manage groups of related components.
   - Light weight Container can be Contained in another Container.

The methods of Component class used in Swing are

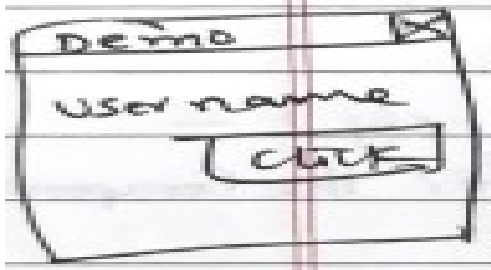
1. public void add(Component c) – add a component on another component (Container)
2. void setSize(int wid, int hgt) – Sets size of Component.
3. void setLayout(LayoutManager obj) – Sets the layout manager for the Component
4. void setVisible(boolean flag) – Sets the visibility of the Component (by default false).

(1)

```
import javax.swing.*;
import java.applet.*;

public class SwingDemo1
{
    SwingDemo1() {
        JFrame jf = new JFrame("Demo");
        jf.setSize(200,300);
        JLabel l1 = new JLable("UserName");
        JButton b1 = new JButton("Click");
        jf.add(l1);
        jf.add(b1);
        jf.setLayout(new FlowLayout());

        jf.setVisible(true);
    }
public static void main(String args[])
    {
        new SwingDemo1();
    }
}
```



Date ___/___/___

If no Layout manager then
b1.SetBounds(x, y, wid, hgt) is used

Sarathi

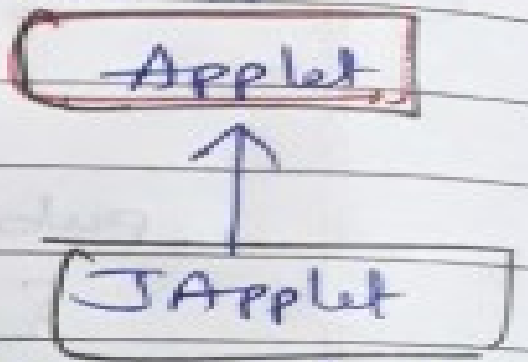
(2)

```
import javax.swing.*;
import javaxapplet.*;

public Class SwingDemo2
{
    public static void main (String args[])
    {
        JFrame jf = new JFrame("Demo");
        jf.setSize(200, 300);
        jf.setLayout(new FlowLayout());
        jf.s
        JLabel l1 = new JLabel("Username");
        JButton b1 = new JButton("Click");

        jf.add(l1);
        jf.add(b1);
        jf.setVisible(true);
    }
}
```

- Swing based applets are similar to AWT-based applets.
- Any Swing applet extends JApplet.
- JApplet defined from Applet, includes the complete functionality of the Applet.




- JApplet is a Top-level Swing Container.
- Swing applets use the four life cycle methods of Applet init(), start(), stop() and destroy().

Eg: SwingDemo3.java

Date ___/___/___

# Swing Components

Saathi

- The following Swing components are lightweight components & derived from JComponent.
- Defined in the package "javax.swing".

## ② JLabel & ImageIcon

- JLabel can be used to display text or an icon.
- It is a passive component
- It does not respond to user input
- The constructors are

```
JLabel ( Icon icon)
JLabel ( String str)
JLabel ( String str, Icon icon, int align)
```

- str & icon are text and icon used for label
- align specifys the horizontal alignment of text in a label. (LEFT, RIGHT, CENTER, LEADING or TRAILING)
- icon is the object of type "Icon".
- An icon is created by using "ImageIcon" class, which implements Icon.

```
ImageIcon( String filename)
```

Page No.

- The methods defined are

```
Icon   getIcon()
String getText()
void   setIcon( Icon icon)
void   setText ( String str)
```

Eg:

```
ImageIcon i1 = new ImageIcon ("filename");
JLabel j1 = new JLabel ("Hello world",
                        i1, JLabel.CENTER);
```

↳ SwingDemo4.java.

## ② JTextField

- Allows to edit one line of text.
- It is the simplest Swing text component.
- The Constructors are

```
JTextField ( int cols)
JTextField ( String str, int cols)
JTextField ( String str)
```

str — Initial string to be displayed
cols — no. of columns in the TextField.

- It generates events in response to user interaction.
- The possible events are
  - actionEvent — when presses Enter
  - CaretEvent — When each time the cursor changes.
    - ↳ defined in "javax.swing.event"
  - TextEvent — when text changes.
- The methods defined are

```
String   getText()
void     setText( String str)
```

Page No.

Eg:- JTextField jt1 = new JTextField("Hello");
JTextField jt2 = new JTextField(15);

③ JButton

- JButton class provides the functionality of a push button
- JButton allows an icon, a string, or both to be associated with the push button.

JButton(Icon icon)
JButton(String str)
JButton(String str, Icon icon)

str and icon → add the string and icon used for the button.

JButton — ActionEvent — actionPerformed()

String getActionCommand() — Returns the string or label of button which is pressed.

void setActionCommand(String str).

Eg: SwingDemo.java

Page No.

**//Program using JLabel , ImageIcon , JTextField , JButton**

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
/*
<APPLET code="SwingDemo4" width=800
height=800>
</APPLET>
*/
public class SwingDemo4 extends JApplet
implements ActionListener
{
JTextField jt1,jt2 ;
JButton jb1,jb2;
JLabel l2;
public void init( )
{
setLayout(new FlowLayout());
ImageIcon i1 = new ImageIcon("demo1.gif" );
JLabel l1 = new JLabel(i1);
l2 = new JLabel("Enter your Name");
jt1 = new JTextField(15);
jt2 = new JTextField(15);
jb1 = new JButton("Click");
jb2 = new JButton(i1);
add(l1); add(l2); add(jt1); add(jb1); add(jt2);
add(jb2);
jb1.addActionListener(this);
jb2.addActionListener(this);
}
public void actionPerformed(ActionEvent ae)
{
jt2.setText(jt1.getText());
repaint();
}
}
```

## ④ JToggleButton

- A toggle button is like a push button. It has two states — pushed and released i.e. When button pressed first time it is stays pressed (not pop up), second time it is pressed, the button is released. (pop up).

- Toggle buttons are objects of JToggleButton.
- The constructor is
  - JToggleButton ( String str)

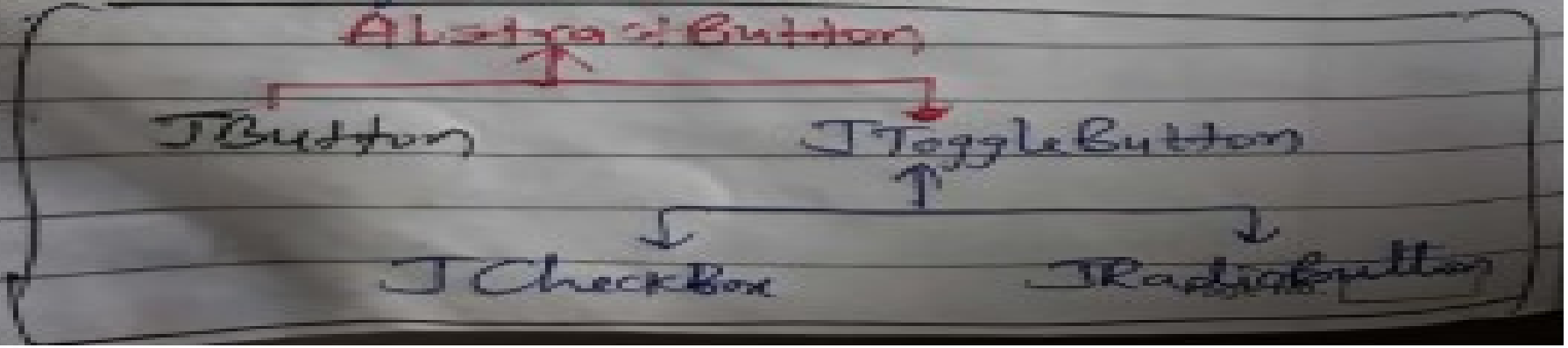
str — String or label on button.

By default the button is in off position.

- JToggleButton ( Image icon)

icon — Image on the button.

Button hierarchy



JToggleButton — ActionEvent
— ItemEvent

For selection ⇒ if pressed in ⇒ on
(Popped out) if pressed out ⇒ OFF.

Eg: "SwingDemos.java"

ActionEvent — actionPerformed()
↳ getActionCommand()
getSource()

ItemEvent — ~~(illegible)~~
itemStateChanged() —
↳ getItem() - returns the item selected (reference)

The method isSelected() returns true if the button is selected.

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
/*
<APPLET code="SwingDemo5" width=300
height=400>
</APPLET>
*/
public class SwingDemo5 extends JApplet
implements ItemListener
{
JLabel jl1,jl2;
JToggleButton jtb1,jtb2;
public void init( )
{
setLayout(new GridLayout(2,2,5,5));
jl1 = new JLabel("->" );
jl2 = new JLabel("->");
jtb1 = new JToggleButton("CSE");
jtb2 = new JToggleButton("ECE");
add(jtb1); add(jl1); add(jtb2); add(jl2);
jtb1.addItemListener(this);
jtb2.addItemListener(this);
}
public void itemStateChanged(ItemEvent ie)
{
if(jtb1.isSelected())
{
jl1.setText("CSE is on");
}
else
{
jl1.setText("CSE is off");
```

## 5. JCheckBox

The JCheckBox class provides the functionality of a check box.

The Constructor is

JCheckBox ( String str)

JCheckBox ( String str, boolean flag)

JCheckBox ( String str, Icon i, boolean flag)

The methods defined are

isSelected() — returns true if the checkbox is selected

getText() — gets the text of the checkbox.

JCheckBox — ItemEvent → itemStateChanged()

— ItemListener

ItemEvent defines getItem() — returning the reference to the item selected.

Eg: JCheckBox c1 = new JCheckBox("C");

SwingDemos.java

Page No.

# 6. JRadioButton

Radio buttons are a group of mutually exclusive buttons, in which only one button can be selected at any one time.

Supported by JRadioButton class.

The Constructors are

JRadioButton (String label)

JRadioButton (String label, boolean on)

JRadioButton (String label, Icon i, boolean on)

The radio buttons must be configured into a group.

A button group is created by the class "ButtonGroup" class.

The elements are added to the group by using add() method

```
JRadioButton b1 = new JRadioButton("My");
ButtonGroup bg = new ButtonGroup();
bg.add(b1);
```

Page No.

ActionEvent — ActionListener — actionPerformed() — getSource()
getActionCommand

ItemEvent — ItemListener — ItemStateChanged() — getItem()

Eg: SwingDemo6.java

```
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
/* <applet code="RadioDemo" width=200 height
=200>
</applet> */
public class RadioDemo extends JApplet
implements ItemListener
{
JCheckBox c1,c2,c3;
JRadioButton r1,r2;
JLabel l1,l2;
String str1="";
String str2="";
public void init()
{
setLayout(new FlowLayout(FlowLayout.LEFT));
l1 = new JLabel(" Courses");
l2 = new JLabel(" Gender");
c1 = new JCheckBox("JAVA ");
c2 = new JCheckBox(".NET ");
c3 = new JCheckBox(" SAP ");
r1= new JRadioButton("Male");
r2= new JRadioButton("FeMale");
ButtonGroup bg = new ButtonGroup();
bg.add(r1); bg.add(r2);
add(c1); add(c2); add(c3); add(r1); add(r2);
add(l1); add(l2);
c1.addItemListener(this); c2.addItemListener(this);
c3.addItemListener(this);
r1.addItemListener(this);
r2.addItemListener(this);
}
public void itemStateChanged(ItemEvent ie)
{
if(c1.isSelected())
str1+= c1.getText();
```

# 7. JComboBox

- Swing provides a combo box which is a combination of a text field and a drop down list, by using JComboBox class.
- A combo box displays one entry and allows user to select a different entry.
- The constructor is ① JComboBox()
  ② JComboBox(Object[] items)
  ↳ An array of items to initialize the combobox.
- To add dynamically items to the list the following method is used.
  void addItem(Object obj)
  void removeItem(int index)
- It generates an ActionEvent when the user selects an item from the list
- It generates an ItemEvent when the state of selection changes, i.e. when an item is selected or deselected.
  Object getSelectedItem() method obtains the item selected in the list.

Page No.

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.event.*;
public class ComboDemo implements
ActionListener
{
JFrame jf;
JPanel jp;
JComboBox jc;
JLabel l1,l2;
String cities[] = {"HYD",
"BNGL","PUNE","DELHI"};
ComboDemo( )
{ jf = new JFrame( ); //Top or High Level
Window
jf.setSize(300,300);
jf.setTitle("Demo");
jp = new JPanel(); // lower or second level
window
l1 = new JLabel("Cities List ");
jp.add(l1);
jp.add(jc);
jp.add(l2);
jf.add(jp);
jf.setVisible(true);
jc.addActionListener (this);
}
public void actionPerformed(ActionEvent
e)
{
String str = (String) jc.getSelectedItem();
l2.setText("--->" +str);
}
public static void main(String args[])
```

# 8. JList

JList supports the selection of one or more items from a list.

It is a list of Strings or any object that can be displayed.

The Constructors are

JList()

JList(Object[] items)

JList generates a ListSelectionEvent when the user makes or changes the selection.

It is handled by ListSelectionListener.

The method implemented is

void valueChanged(ListSelectionEvent e)

This event is defined in Javax.swing.event

By default the JList allows to select multiple ranges of items within the list.

The methods defined in JList are

P.T.O.

page no.

- void setSelectionMode( int mode)
  - mode can take value
    - SINGLE_SELECTION
    - SINGLE_INTERVAL_SELECTION
    - MULTIPLE_INTERVAL_SELECTION
- int getSelectedIndex() — returns the index, if no item selected return
- Object getSelectedValue() — returns the selected value from the list.

```
//Swing JList Demo
import javax.swing.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;
/*
<applet code="ListDemo" width=300
height=300>
</applet>
*/
public class ListDemo extends JApplet
implements ActionListener,
ListSelectionListener
{
JList jl ;
JButton jb;
JTextArea jta;
String names[] = {"C " , "C++" , "JAVA"};
public void init()
{
jl = new JList(names);
jb = new JButton("Language");
jta = new JTextArea(10,20);
setLayout(new FlowLayout());
add(jb);
add(jl);
add(jta);
jb.addActionListener(this);
jl.addListSelectionListener(this);
}
public void actionPerformed(ActionEvent
ae)
{
String str =(String) jl.getSelectedValue();
jta.setText(str);
}
public void
```

10. JScrollPane

- JScrollPane is a lightweight container that automatically handles the scrolling of another component.
- The component scrolled can be Table, a group of controls in a light weight container like JPanel.
- Horizontal or vertical & both scroll bars are automatically provided.
- Eliminates the use of individual scroll bars
- The Constructors are

```
JScrollPane ( Component comp)
JScrollPane ( )
```

comp → is the component to be scrolled

- The Procedure is
  1. Create the Component to be scrolled
  2. Create an instance of JScrollPane, passing component to it
  3. Add the scroll pane to Content Pane

Page No.

# 9. JTable

- JTable component displays rows & columns of data.
- Allows to drag a column to a new position.
- It is also possible to select a row, column or cell with in the table.
- Allows to change data in the cell.
- JTable contains one or more columns of information at the top each column is having a headings.
- The Constructor is

  JTable ( Object data[][], Object head[])
  - data is actual data
  - head is headings

- JTable generates two frequent events
  1. ListSelectionEvent — generates when the user selects something in table. Allows to select multiple rows by default we can also select multiple columns
  2. TableModelEvent — generates when table data changed.

Page No.

The Simple JTable creation is

1. Create an instance of JTable
2. Create a JScrollPane object, to Scroll in the table
3. Add the table to ScrollPane
4. Add the scroll pane to Content Pane.

Eg: SwingDemo9.java

```
//Simple Swing JTable
import javax.swing.*;
import java.awt.*;
public class TableDemo
{
JFrame jf;
JScrollPane js;
JTable jt;
String data[][] = { {"501" ,
"Abhishek" ,"80"} , {"509" , "
Karimnagar" ,"90"},
{"519" ,"Sai Kiran" , "70"}, {"519" ,"Sai
Kiran" , "70"},
{"519" ,"Sai Kiran" , "70"};
String head[ ] = { " No " , "Name" ,
"Percent"};
TableDemo( )
{
jf = new JFrame( ); //Top or High Level
Window
jf.setSize(150,150);
jf.setTitle("Table Demo");
jt = new JTable(data , head);
js = new JScrollPane(jt);
jf.add(js); setVisible(true);
}

public static void main(String args[])
{
new TableDemo();
}
}
```

# 12. JTabbedPane

- It manages a set of components by linking them with the tabs.
- Selecting a tab causes the component associated with that tab to come to the forefront.
- JTabbedPane class is used to create a tabbed pane.
- The Constructor is

```
JTabbePane ( )
```

- It is a Single selection model.
- The tabs are added by using addTab() method.

```
void addTab (String name,
                    Component comp);
```

name — name for the tab
comp — Component to be added to tab.

- Each tab hold a set of Components.

The general Procedure is

1. Create an instance of JTabbedPane
2. Add each tab by calling addTab()
3. Add the tabbed pane to the Content pane.

Eg: TabbedPaneDemo.

```
// Demonstrate JTabbedPane.
import javax.swing.*;
/* <applet code="TabbedPane" width=400
height=100>
</applet> */
public class TabbedPane extends JApplet
{
JTabbedPane jtp;
public void init()
{
jtp = new JTabbedPane();
jtp.addTab("CSE", new CSEPanel());
jtp.addTab("ECE", new ECEPanel());
jtp.addTab("EEE", new EEEPanel());
add(jtp);
}
}
// Make the panels that will be added to the
tabbed pane.
class CSEPanel extends JPanel
{
public CSEPanel()
{
JButton b1 = new JButton("Computer
Fundamentals");
add(b1);
JButton b2 = new JButton("Computer
Organization");
add(b2);
JButton b3 = new JButton("Computer
Graphics");
```

```
class EEEPanel extends JPanel
{
public EEEPanel()
{
JComboBox<String> jcb = new
JComboBox<String>();
jcb.addItem("ELECTRICAL MACHINES");
jcb.addItem("POWER SYSTEMS");
jcb.addItem("CONTROL SYSTEMS");
add(jcb);
}
}
```

## 13. JTextArea

It is a multiline region that displays text in the component.

It allows editing of multiple line text.

The Constructors are

```
JTextArea()
JTextArea(String str)
JTextArea(int row, int col)
JTextArea(String str, int row, int col)
```

str – Initial text to be displayed

row & col – no. of rows & cols of the control

The methods defined are

```
void setRows(int rows)
void setColumns(int cols)
void setFont(Font f)
void insert(String s, int position)
void append(String s)
String getText()
void setText(String str)
```

Eg: `JTextArea ta1 = new JTextArea(10, 20);`

Page No.

```
/* Create a user interface to insert
employee details, Display the data in
Text area. */
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.event.*;
public class EmpDemo implements
ActionListener
{
String data="EMP:" ;
JFrame jf;
JPanel jp;
JLabel l1,l2,l3,l4,l5,l6;
JTextField t1,t2,t3;
JRadioButton r1,r2;
ButtonGroup bg;
JComboBox jc;
JCheckBox c1,c2,c3;
JButton b1;
JTextArea ta1;
String cities[] = {"HYD", "BNGL","PUNE","DELHI"};
jf = new JFrame( ); //Top or High Level Window
jf.setSize(300,300);
jf.setTitle("Demo");
jp = new JPanel(); // lower or second level window
EmpDemo( )
{
l1 = new JLabel("EMP-ID"); l2 = new
JLabel("EMP-NAME");
l3 = new JLabel("Designation");
l4 = new JLabel("Gender");
l5 = new JLabel("CITY"); l6= new
JLabel("COMPANY");
```

```
public void actionPerformed(ActionEvent e)
{
data= data + t1.getText() + ":";
data= data + t2.getText() + ":";
data= data + t3.getText() + ":";
if(r1.isSelected())
data= data+ "Male" +";";
else
data= data+ "FeMale" +";";
String str = (String) jc.getSelectedItem();
data = data + str + ":";
if(c1.isSelected())
data= data + c1.getText();
else if(c2.isSelected())
data=data +c2.getText();
else
data=data+c3.getText();
ta1.setText(data);
}
public static void main(String args[])
{
new EmpDemo(); } }
```

```
/* Create a user interface to USE
JTable
import javax.swing.JFrame;
import javax.swing.JScrollPane;
import javax.swing.JTable;
/* import javax.swing.*/
public class JStudent
{
JFrame f;
JTable j;
JStudent()
{
f = new JFrame();
f.setTitle("JTable Example");
String[][] data = {
{ "101", "Rajesh", "CSE","II","78.5"},
{ "102", "Harsha", "CSE","II","87.5"},
{ "103", "Varsha", "CSE","II","65.5"},
{ "104", "Kiran", "IT","II","75.5"},
{ "105", "Karan", "IT","II","87.5"},
};
// Column Names
String[] head = { "RollNo", "Name",
"Department","Branch" , "Percentage"};
// Initializing the JTable
j = new JTable(data, head);
j.setBounds(30, 40, 200, 300);
// adding it to JScrollPane
JScrollPane sp = new JScrollPane(j);
f.add(sp);
// Frame Size
f.setSize(500, 200);
// Frame Visible = true
f.setVisible(true);
}
```